Ονοματεπώνυμο: Παναγιώτης Ντενέζος

A.M.: 5853

email: ntenezos@ceid.upatras.gr

Υλοποιήθηκαν όλα τα ερωτήματα της άσκησης.

➢ Client (client.c)

Το πρόγραμμα για τον client περνάει με επιτυχία τον αυτόματο έλεγχο.

Το πρόγραμμα αποτελείται από δυο συναρτήσεις:

- την *writen*
  Η *writen* χρησιμοποιείται αντί της *write*. Ο λόγος είναι ότι η συνάρτηση *writen* στέλνει μέσα από το socket το μήνυμα (με χρήση της write) και ελέγχει αν έχουν σταλεί όλα τα bytes της πληροφορίας. Αυτή η διαδικασία γίνεται επαναληπτικά μέχρι να μην υπάρχει άλλο byte πληροφορίας. Με αποτέλεσμα να μην χάνεται κάποια πληροφορία.
- την *main*
  Αρχικά ο client δέχεται σαν πρώτες παραμέτρους τον server που θα συνδεθεί και το port. Ξεκινώντας η *main*, δημιουργούνται οι απαραίτητες προϋποθέσεις για να εδραιωθεί η σύνδεση μεταξύ του client και του server.. Αφού γίνει αυτό ξεκινάει μια for loop με βάση το argc έτσι ώστε να ξεκινήσει το parsing των command line arguments. Για κάθε ένα argument ελέγχει αν είναι put ή get. Αν είναι put ελέγχει αν υπάρχουν δυο ακόμα arguments (key και value). Στην συνέχεια, περνάει σε έναν buffer την εντολή σε μορφή p<key>\0<value>\0 και την στέλνει στον server. Αν είναι get ελέγχει

αν υπάρχει ένα ακόμα argument (key) και μετά περνάει στο buffer την εντολή σε μορφή g<key>\0 και την στέλνει στον server. Μόλις ο server απαντήσει μέσω της συνάρτησης *read* ο client διαβάζει την απάντηση. Η ανάγνωση της απάντησης γίνεται επαναληπτικά με έναν temporary buffer ώστε να είναι σίγουρο ότι δεν θα χαθεί κάποιο byte πληροφορίας (στην ίδια λογική με την *written)*. Έπειτα, ελέγχει το πρώτο γράμμα της απάντησης που δέχθηκε και αν είναι f (found) τότε την εκτυπώνει αλλιώς αν είναι n (not found) τότε εκτυπώνει απλά μια κενή γραμμή. Τέλος, στην περίπτωση που το argument δεν είναι ούτε put ούτε get ο client στέλνει ένα null char στο server, σπάει το for loop και τερματίζει.

- **Iterative server (serv1.c)**

Το πρόγραμμα για τον serv1 περνάει με επιτυχία τον αυτόματο έλεγχο.

Το key-value store του server υλοποιήθηκε με έναν δισδιάστατο πίνακα. Χρησιμοποιείται global counter για να γνωρίζουμε ανά πάσα στιγμή το πόσα ζευγάρια περιέχει το key-value store του server.
Ακόμα, χρησιμοποιείται μια global μεταβλητή sync_flag, η οποία σε κάθε εντολή που δέχεται ο server παίρνει την τιμή 0 έτσι ώστε μόλις ο server κάνει το parse της εντολής μπαίνει σε loop μέχρι να ολοκληρωθεί η εκτέλεση της. Έτσι εξασφαλίζεται ότι όλες οι εντολές θα εκτελεσθούν με την σειρά που έρχονται.

Το πρόγραμμα αποτελείται από τέσσερεις συναρτήσεις:

- την *writen* (αναφέρεται παραπάνω)
- την *put*

  Η *put* δέχεται σαν ορίσματα δυο strings, ένα key και ένα value. Η συνάρτηση αυτή μέσα από μια for loop με βάση το πλήθος των εντολών put που έχουν δοθεί ελέγχει αν υπάρχει καταχωρημένο το κλειδί (key) που στάλθηκε και αν υπάρχει απλά ανανεώνει την αντίστοιχη τιμή για το value, αλλιώς προσθέτει στην μνήμη ένα ακόμα

ζευγάρι key-value και αυξάνει τον global counter. Τέλος, κάνει το sync_flag 1 για τους λόγους που αναφέρθηκαν παραπάνω.

- την *get*

  Η *get* δέχεται σαν όρισμα ένα string key. Η συνάρτηση αυτή μέσα από μια for loop με βάση το πλήθος των εντολών put που έχουν δοθεί ελέγχει αν υπάρχει καταχωρημένο το κλειδί (key) που στάλθηκε και αν υπάρχει επιστρέφει το αντίστοιχο value, αλλιώς επιστρέφει το αλφαριθμητικό NULL. Τέλος, κάνει το sync_flag 1 για τους λόγους που αναφέρθηκαν παραπάνω.

  Ο λόγος που στέλνετε το NULL σαν αλφαριθμητικό και όχι σαν null pointer είναι για λόγους ευκολίας στην σύγκριση που γίνεται μετά στην main.

- την *main*

  Αρχικά ο server δέχεται σαν παράμετρο το port στο οποίο θα γίνει η σύνδεση. Αν δεν δοθεί από το χρήστη το port τότε by default το port είναι 1234. Ξεκινώντας η *main,* δημιουργούνται οι απαραίτητες προϋποθέσεις ώστε να μπορεί να συνδεθεί κάποιος client. Στην συνέχεια ο server μπαίνει σε μια infinite loop while(1) και περιμένει μέχρι να του στείλει κάτι κάποιος client. Μόλις ο server δεχθεί το μήνυμα, το κάνει parse χαρακτήρα χαρακτήρα και ανάλογα με το είδος της εντολής put η get. Αν ο πρώτος χαρακτήρας του μηνύματος είναι p τότε η εντολή είναι put και ο server αποθηκεύει κάθε επόμενο χαρακτήρα μέχρι να συναντήσει τον null character σε έναν μονοδιάστατο πίνακα key και όλους τους επόμενους μέχρι το επόμενο null character σε έναν μονοδιάστατο πίνακα value και στην συνέχεια τα στέλνει στην συνάρτηση *put.* Αν, όμως, είναι g τότε τότε η εντολή είναι get και ο server επαναλαμβάνει την ίδια διαδικασία με την εντολή put αλλά μόνο για το key και μετά το στέλνει στην συνάρτηση *get*. Μόλις η *get* επιστρέψει το αποτέλεσμα ο server κάνει την σύγκριση με το αλφαριθμητικό NULL. Αν είναι αλήθεια, δηλαδή δεν βρέθηκε το κλειδί τότε στέλνει στον client το χαρακτήρα n, αλλιώς παίρνει το επιστρεφόμενο αλφαριθμητικό, του βάζει μπροστά το χαρακτήρα f και το στέλνει στον client. Τέλος, αν ο server δεχτεί κάτι

που δεν ξεκινάει από p ή g κλείνει την σύνδεση μεταξύ αυτού και του client.

- **<u>On-demand forking Server (serv2.c)</u>**

  Το πρόγραμμα για τον serv2 περνάει με επιτυχία τον αυτόματο έλεγχο εκτός του test 10.

  Για τον serv2 ισχύουν ότι και στον serv1 και κάποια ακόμα επιπλέον. Επειδή θέλουμε όλα τα process να μπορούν να χρησιμοποιήσουν τον ίδιο χώρο μνήμης, γίνεται χρήση ενός struct, το οποίο χρησιμοποιείται για αυτό το λόγο.

  Το πρόγραμμα αποτελείται από έξι συναρτήσεις:

  - Την *sig_chld*

    Η συνάρτηση *sig_chld* είναι ένας signal handler. Ουσιαστικά, χρησιμοποιείται έτσι ώστε ο πατέρας να περιμένει να τερματίσουν πρώτα όλα τα παιδιά/processes. Έτσι αποφεύγουμε την δημιουργία zombies processes.
  - την *writen* (αναφέρεται παραπάνω)
  - την *my_func*

    Η συνάρτηση *my_func* περιέχει όλο το λειτουργικό κομμάτι του serv1. Ο λόγος που δημιουργήθηκε είναι καθαρά για λόγους απλότητας της *main* και πιο εύκολης οπτικοποίησης/δομής του κώδικα.
  - την *put*

    Η συνάρτηση *put* χρησιμοποιείται ακριβώς όπως στον serv1 με την μόνη διαφορά τώρα ότι δημιουργείται εσωτερικά η δομή my_mem έτσι ώστε να μπορεί να έχει πρόσβαση στην κοινή μνήμη.
  - την *get*

    Ισχύει ότι και στην συνάρτηση *put.*
  - την *main*

    Όλα τα πρώτα βήματα είναι ίδια με την συνάρτηση *main* του serv1. Για το serv2 δημιουργείται η δομή my_mem για να έχει πρόσβαση στην κοινή μνήμη. Στην συνέχεια, μόλις μπει στο while loop και

κάποιος client συνδεθεί μέσω της συνάρτησης *fork* ξεκινάει ένα καινούργιο process και με χρήση σεμαφόρων προστατεύει την κρίσιμη περιοχή κατεβάζοντας την σεμαφόρο. Με αυτόν τον τρόπο αποφεύγονται τα race conditions.

- **Preforking Server (serv3.c)**

Το πρόγραμμα για τον serv3 περνάει με επιτυχία τον αυτόματο έλεγχο εκτός του test 10.

Για τον serv3 ισχύουν ότι και στον serv2.

Το πρόγραμμα αποτελείται από έξι συναρτήσεις:

- Την *sig_chld* (αναφέρεται παραπάνω)
- την *writen* (αναφέρεται παραπάνω)
- την *my_func* (αναφέρεται παραπάνω)
- την *put* (αναφέρεται παραπάνω)
- την *get* (αναφέρεται παραπάνω)
- την *main*

  Αρχικά ο server δέχεται σαν παράμετρο εκτός από το port στο οποίο θα γίνει η σύνδεση και τον αριθμό των processes που θα δημιουργήσει ο server. Η μόνη διαφορά με τον serv2 είναι ότι το while loop βρίσκεται τώρα μέσα σε ένα for loop με βάση τον προκαθορισμένο αριθμό των processes. Με αυτόν τον τρόπο μπορούν να συνδεθούν μόνο τόσοι clients όσοι και τα processes.

- **Multi-treaded server (serv4.c)**

Το πρόγραμμα για τον serv4 περνάει με επιτυχία τον αυτόματο έλεγχο εκτός των test 10, 12 και 14.

Για τον serv4 ισχύουν ότι και στον serv2 μόνο που εδώ δεν χρησιμοποιούμε κοινή μνήμη, καθώς είναι γνωστό ότι τα threads έχουν κοινό χώρο διευθύνσεων, αυτόν του πατέρα. Αντί για δομή έχουμε global μεταβλητές. Γίνεται πάλι χρήση της μεταβλητής sync_flag.

Το πρόγραμμα αποτελείται από έξι συναρτήσεις:

- Την *sig_chld* (αναφέρεται παραπάνω)
- την *writen* (αναφέρεται παραπάνω)
- την *my_func* (αναφέρεται παραπάνω)
- την *put* (αναφέρεται παραπάνω)
- την *get* (αναφέρεται παραπάνω)
- την *main*

  Αρχικά ο server δέχεται σαν παράμετρο μόνο το port στο οποίο θα γίνει η σύνδεση. Η διαφορά με τον serv2 είναι ότι επειδή αντί για processes έχουμε treads δεν υπάρχουν σεμαφόροι. Συνεπώς, μέσα στο while loop δημιουργούμε τα threads και στην κρίσιμη περιοχή η *my_func* δέχεται το socket του κάθε client με αναφορά. Τέλος, ο πατέρας για να κλείσει περιμένει πρώτα να κλείσουν όλα τα threads που έχουν δημιουργηθεί.